Data Cleaning Procedure

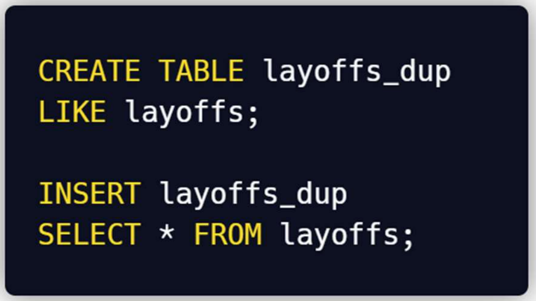
Overview:

- Analyses a dataset of **2,500 records** capturing layoffs from **2,000 companies** across **60 countries**.
- Key columns: *company, location, industry, total_laid_off, percentage_laid_off, date, stage, country, and funds_raised_millions*.
- Transforms raw, messy data into a polished and analysis-ready dataset, ensuring it's clean, consistent, and reliable for future insights.
- Ensures data accuracy and readiness for analysis through **4-part cleaning process**:
    1. **Remove Duplicates**
    2. **Standardize the Data** (spell-checking)
    3. **Handle NULL/Blank Values**
    4. **Delete Unnecessary Data**

1. *Remove Duplicates:*

i. Create a duplicate table:

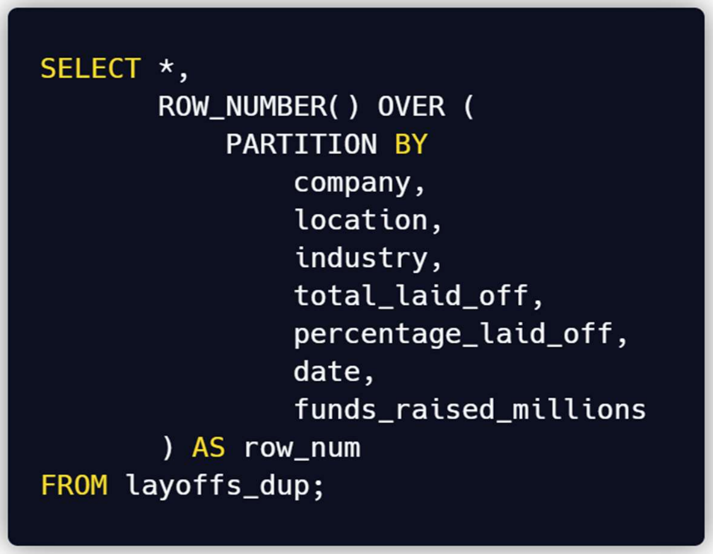
- Preserves Data Integrity: Keeps the original data safe for reference or rollback.
- Enables Traceability: Facilitates comparison between cleaned and raw data.
- Supports Experimentation: Allows testing cleaning methods without risking the source data.
- Optimizes Performance: Refined data can be indexed for faster queries.



```
CREATE TABLE layoffs_dup
LIKE layoffs;

INSERT layoffs_dup
SELECT * FROM layoffs;
```

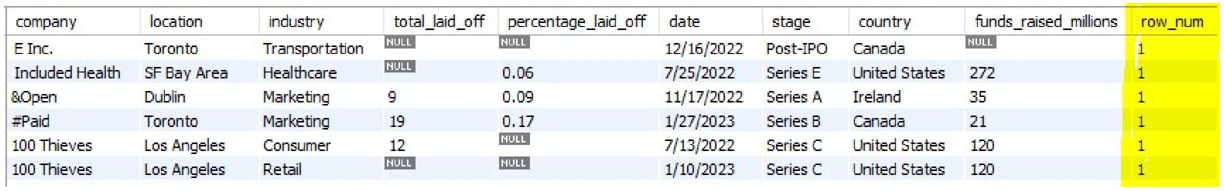
ii. Deleting Duplicate Records:

- Creating a 'duplicate indicator' column:
  - To delete duplicate records, we first have to create a new column that acts as an indicator if a record is duplicate or not.
  - We can do that with the window function **ROW_NUMBER ()**



```
SELECT *,
       ROW_NUMBER( ) OVER (
           PARTITION BY
               company,
               location,
               industry,
               total_laid_off,
               percentage_laid_off,
               date,
               funds_raised_millions
       ) AS row_num
FROM layoffs_dup;
```

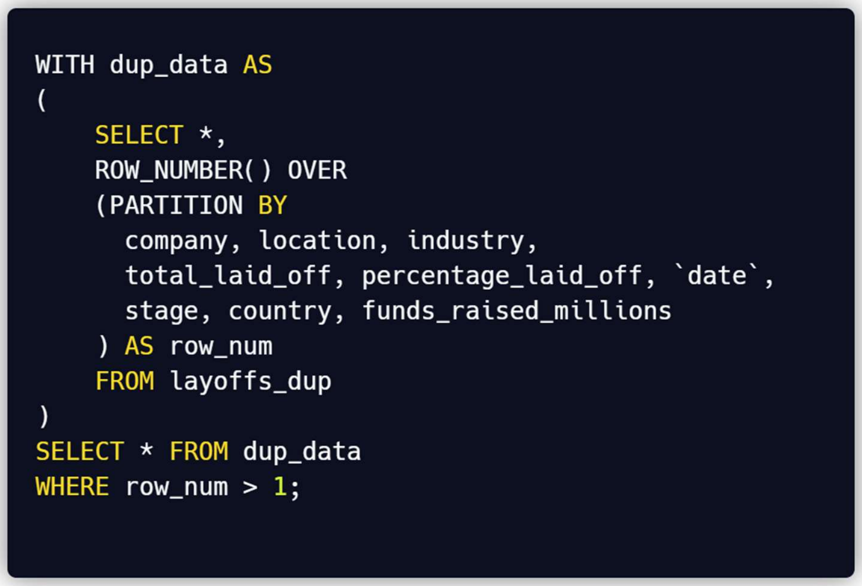
- This creates a separate column which identifies unique rows by assigning them with “1” and duplicates with values greater than “1”



company	location	industry	total_laid_off	percentage_laid_off	date	stage	country	funds_raised_millions	row_num
E Inc.	Toronto	Transportation	NULL	NULL	12/16/2022	Post-IPO	Canada	NULL	1
Included Health	SF Bay Area	Healthcare	NULL	0.06	7/25/2022	Series E	United States	272	1
&Open	Dublin	Marketing	9	0.09	11/17/2022	Series A	Ireland	35	1
#Paid	Toronto	Marketing	19	0.17	1/27/2023	Series B	Canada	21	1
100 Thieves	Los Angeles	Consumer	12	NULL	7/13/2022	Series C	United States	120	1
100 Thieves	Los Angeles	Retail	NULL	NULL	1/10/2023	Series C	United States	120	1

iii. Use ROW_NUMBER () to identify duplicates:

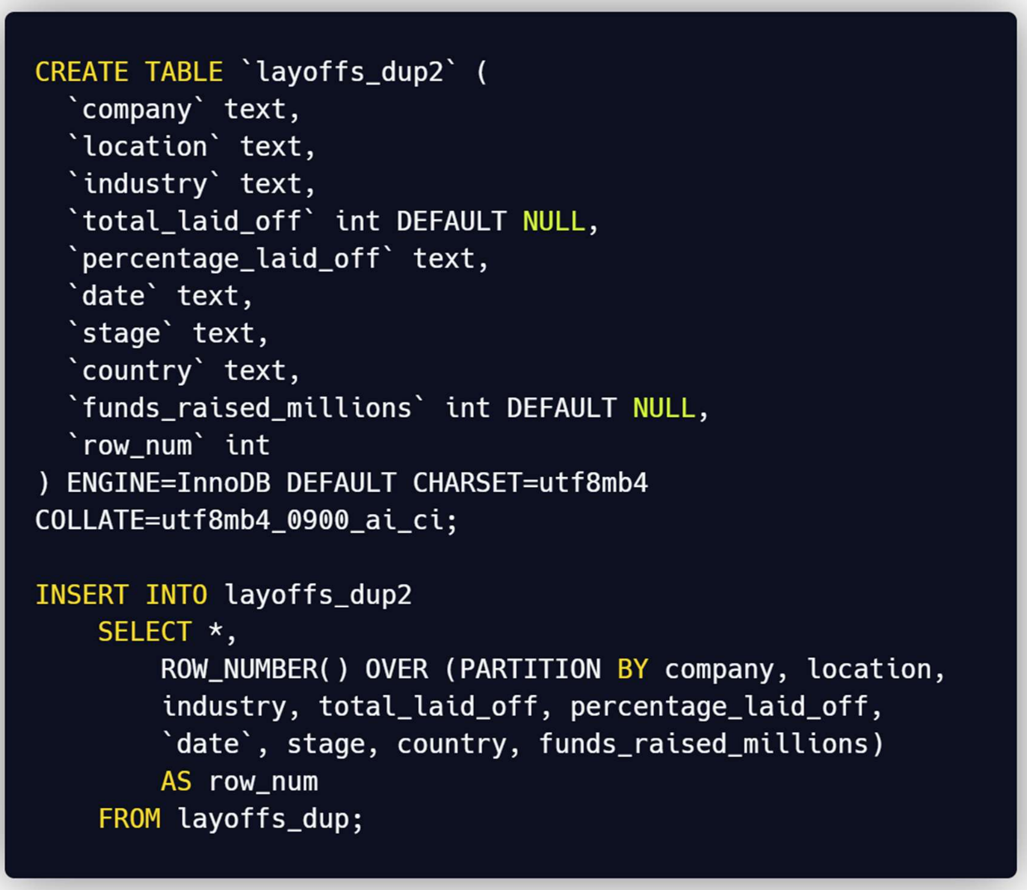
Modify the query to find records whose “row_num” is greater 1 (Use either Subqueries or Common Table Expressions (CTEs)):



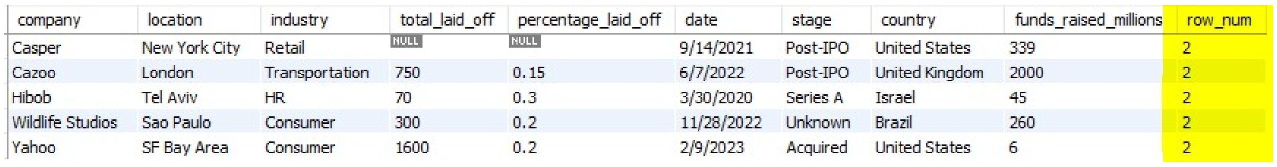
```
WITH dup_data AS
(
    SELECT *,
    ROW_NUMBER() OVER
    (PARTITION BY
      company, location, industry,
      total_laid_off, percentage_laid_off, `date`,
      stage, country, funds_raised_millions
    ) AS row_num
    FROM layoffs_dup
)
SELECT * FROM dup_data
WHERE row_num > 1;
```

iv. Create another duplicate table for safe deletion of duplicate records:

- Since this step involves deletion of data, create another table to ensure safe deletion of files
- Add the ‘*layoffs_dup’* data into the new table and rename it ‘***layoffs_dup2’***

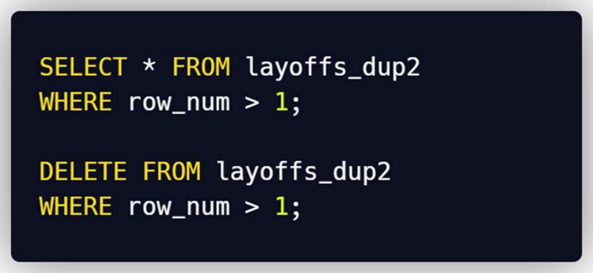


```sql
CREATE TABLE `layoffs_dup2` (
  `company` text,
  `location` text,
  `industry` text,
  `total_laid_off` int DEFAULT NULL,
  `percentage_laid_off` text,
  `date` text,
  `stage` text,
  `country` text,
  `funds_raised_millions` int DEFAULT NULL,
  `row_num` int
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;

INSERT INTO layoffs_dup2
    SELECT *,
        ROW_NUMBER() OVER (PARTITION BY company, location,
        industry, total_laid_off, percentage_laid_off,
        `date`, stage, country, funds_raised_millions)
        AS row_num
    FROM layoffs_dup;
```



company	location	industry	total_laid_off	percentage_laid_off	date	stage	country	funds_raised_millions	row_num
Casper	New York City	Retail	NULL	NULL	9/14/2021	Post-IPO	United States	339	2
Cazoo	London	Transportation	750	0.15	6/7/2022	Post-IPO	United Kingdom	2000	2
Hibob	Tel Aviv	HR	70	0.3	3/30/2020	Series A	Israel	45	2
Wildlife Studios	Sao Paulo	Consumer	300	0.2	11/28/2022	Unknown	Brazil	260	2
Yahoo	SF Bay Area	Consumer	1600	0.2	2/9/2023	Acquired	United States	6	2

- Execute the same queries to find the duplicate records and delete those records from *'layoffs_dup2'*

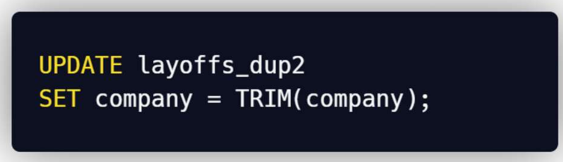


```sql
SELECT * FROM layoffs_dup2
WHERE row_num > 1;

DELETE FROM layoffs_dup2
WHERE row_num > 1;
```

2. *Standardizing the data:*

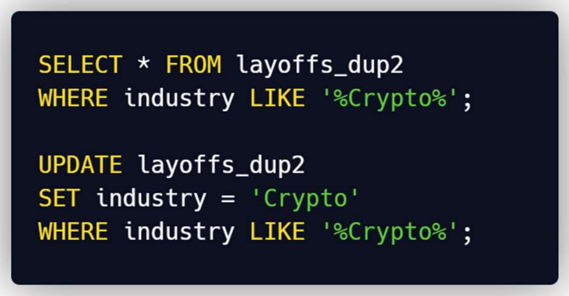
i. Use TRIM () Function to remove the gaps before the names:

- TRIM () function removes spaces present before and after a company name



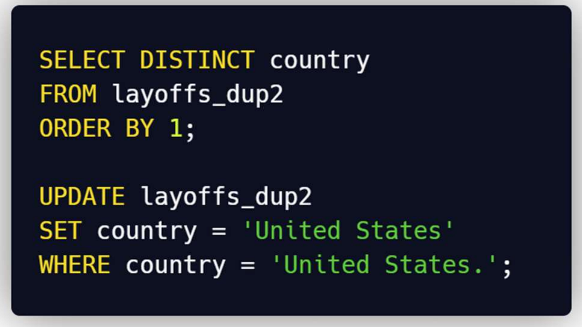
```sql
UPDATE layoffs_dup2
SET company = TRIM(company);
```

ii. Merge similarly named industries like ' crypto', 'crypto currency', 'crypto-currency':



```sql
SELECT * FROM layoffs_dup2
WHERE industry LIKE '%Crypto%';

UPDATE layoffs_dup2
SET industry = 'Crypto'
WHERE industry LIKE '%Crypto%';
```

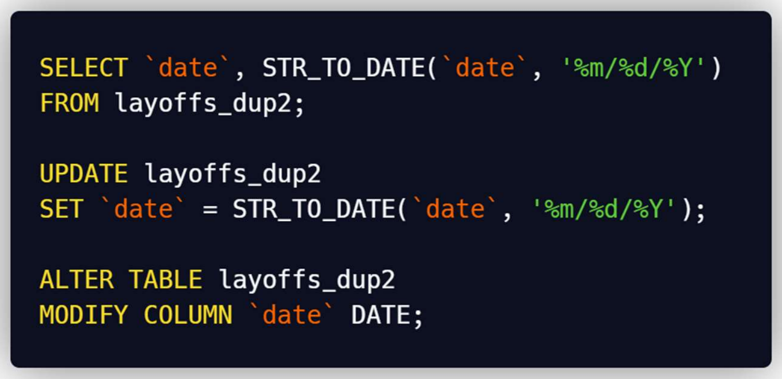
iii. Look for more unclean data in ‘countries’ columns with DISTINCT ():



```sql
SELECT DISTINCT country
FROM layoffs_dup2
ORDER BY 1;

UPDATE layoffs_dup2
SET country = 'United States'
WHERE country = 'United States.';
```

iv. Change formats of some columns like ‘date’:

- First convert the text in the ‘date’ column into data datatype with the help of **‘STR_TO_DATE ()’** function
- Then update the column values with the same **‘STR_TO_DATE ()’**
- Finally, change the datatype of the ‘date’ column from text to date datatype:

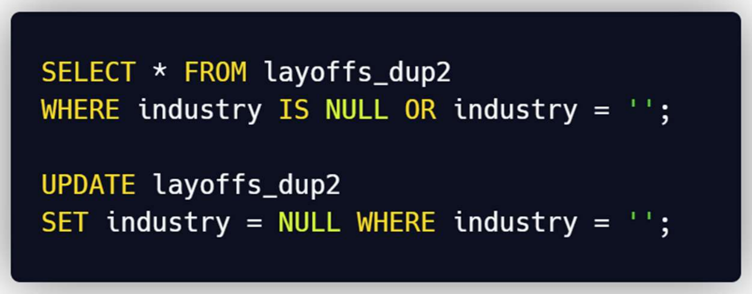


```sql
SELECT `date`, STR_TO_DATE(`date`, '%m/%d/%Y')
FROM layoffs_dup2;

UPDATE layoffs_dup2
SET `date` = STR_TO_DATE(`date`, '%m/%d/%Y');

ALTER TABLE layoffs_dup2
MODIFY COLUMN `date` DATE;
```

---

3. *Dealing with NULL values*

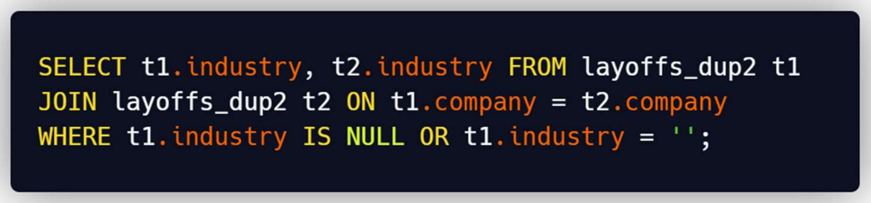
i. Converting blanks into null values for consistency:



```sql
SELECT * FROM layoffs_dup2
WHERE industry IS NULL OR industry = '';

UPDATE layoffs_dup2
SET industry = NULL WHERE industry = '';
```

ii. Check for Possible Replacements:

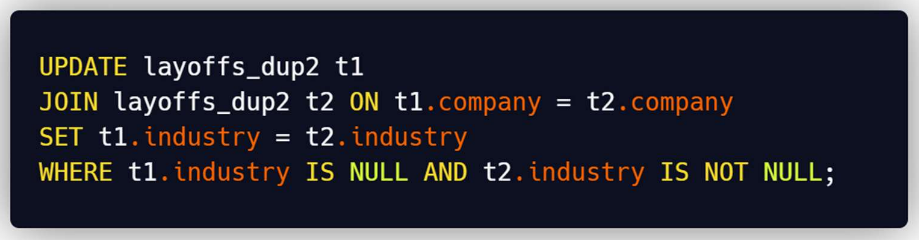
- Find existing values for the same company to use as replacements



```sql
SELECT t1.industry, t2.industry FROM layoffs_dup2 t1
JOIN layoffs_dup2 t2 ON t1.company = t2.company
WHERE t1.industry IS NULL OR t1.industry = '';
```

iii. Fill missing values:

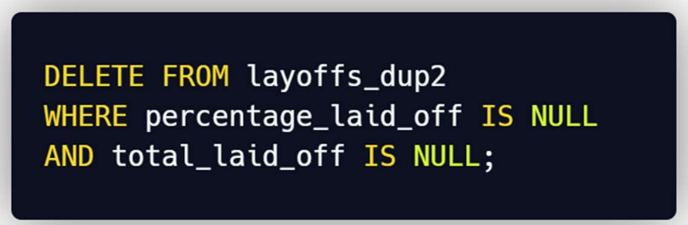
- Update NULL industries using available values from the same company



```sql
UPDATE layoffs_dup2 t1
JOIN layoffs_dup2 t2 ON t1.company = t2.company
SET t1.industry = t2.industry
WHERE t1.industry IS NULL AND t2.industry IS NOT NULL;
```

4. *Deleting unnecessary data:*

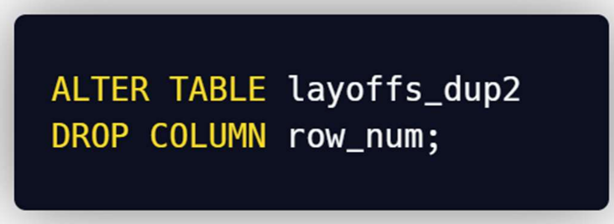
i. Deleting records that don’t play a big role in analysis:



```
DELETE FROM layoffs_dup2
WHERE percentage_laid_off IS NULL
AND total_laid_off IS NULL;
```

ii. Dropping the ‘row_num’ column:

- Row_num has been used just to identify the duplicate records, which isn’t of a big use



```
ALTER TABLE layoffs_dup2
DROP COLUMN row_num;
```